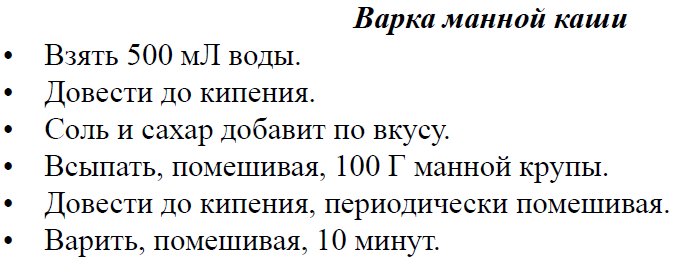
**ТЕОРИЯ АЛГОРИТМОВ**

**Экзаменационные вопросы**

**ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ ТЕОРИИ АЛГОРИТМОВ**

1. **Представление алгоритмов на интуитивном уровне**

*Алгоритм* − набор систематических инструкций по выполнению в строго определённом порядке необходимых действий для решения всех задач какого-либо заданного класса (неформальное определение).



1. **Определения понятия “алгоритм”**

*Алгоритм* − это заданное на некотором языке конечное предписание, задающее конечную последовательность выполнимых элементарных операций для решения задачи, общее для класса возможных исходных данных (формализированное определение).

*Алгоритм* − это всякая система вычислений, выполняемых по строго определённым правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи (А.Н. Колмогоров).

*Алгоритм* − это точное предписание, определяющее вычислительный процесс, идущий от варьируемых исходных данных к искомому результату (А.А. Марков).

Пусть D − множество исходных данных задачи, а R − множество возможных результатов, тогда мы можем говорить, что алгоритм осуществляет отображение D→R.

* частичный, если выполняется для ;
* полный, если выполняется для ;

*Алгоритмическая система* − общий способ задания алгоритма.

Требования к алгоритму:

* конечность записи (должен содержать конечное количество элементарно выполнимых предписаний);
* конечность действий (должен выполнять конечное количество шагов при решении задачи);
* универсальность (должен быть единым для всех допустимых исходных данных);
* правильность (должен приводить к правильному по отношению к поставленной задаче решению);

1. **Основные разделы теории алгоритмов**
2. Общие понятия и определения теории алгоритмов

Например: конструктивный объект, слова в алфавите, матрицы, вектора, массивы, деревья, комплексы (графы), ансамбли, итеративный процесс, локальная операция.

1. Теория вычислительных моделей

Служит базой для построения архитектуры вычислительных систем.

* машины А. Тьюринга;
* машины Э. Поста;
* машины А.Н. Колмогорова;
* машины А.А. Маркова;

1. Теория исчислений

*Исчисление* − это список порождающих (разрешающих) правил или правил вывода, позволяющая осуществлять переходы от одного конструктивного объекта к другим конструктивным объектам.

1. Исследование μ-рекурсивных функций

μ-рекурсивная функция совпадает по смыслу с вычислимой теоретико-числовой функцией, которая определяется как числовая функция, получаемая из фиксированного набора простейших исходных функций с помощью применения в произвольном порядке простейших операций, так же выбираемых из некого фиксированного набора.

1. Изучение проблемы сводимости Поста

Сводимость проблемы В к проблеме А означает, по Посту, наличие некоторого способа, который позволяет эффективно ответить на вопрос «», пользуясь ответом «».

1. Исследование вычислимых операций

Изучение специальных способов задания операций и схем программ.

1. Изучение программ как объекта вычисления и порождения

* способы программирования;
* универсальные алгоритмы и универсальные функции;
* гёделевские модели;
* вычислительные структуры;

1. **Математические приложения теории алгоритмов**
2. Исследование массовых проблем

*Алгоритмическая проблема* − построение алгоритма с заданными свойствами.

*Единичная проблема* состоит в требовании предъявить объект, который удовлетворяет определённым условиям, называемый решением проблемы.

*Массовая проблема* состоит в серии (часто бесконечной) единичных проблем и требовании решить все эти проблемы.

1. Конструктивная семантика - лежит в основании теории компиляции программ.
2. Анализ формализованных языков логики и арифметики - лежит в основании предикатных языков.
3. Вычислительный анализ
4. Нумерованные структуры
5. Построение псевдослучайных последовательностей
6. Алгоритмический подход к понятию количества информации
7. Оценка сложности решения отдельных задач
8. Влияние на алгоритмическую практику

* алгоритмы и возможности формализации;
* неалгоритмическое описание вычислимых функций (непроцедурные ЛИСП);
* вычислительные и порождающие модели (конечные автоматы и формальные грамматики

1. **Современные направления теории алгоритмов**
2. Классическая теория алгоритмов − изучает свойства алгоритмов и их формальное описание, описывает, как алгоритмы должны быть написаны и как они могут быть оптимизированы, чтобы решать задачи более эффективно.

* формулировка задач в терминах формальных языков;
* понятие задачи разрешения, введение сложностных классов;
* исследование класса NP полных задач;

1. Теория асимптотического анализа алгоритмов − изучает поведение алгоритмов при увеличении размера входных данных, позволяет определить время выполнения алгоритма в зависимости от размера входных данных.

* критерии оценки алгоритмов;
* методы получения асимптотических оценок;
* асимптотический анализ трудоёмкости или времени выполнения

1. Теория практического анализа вычислительных алгоритмов − изучает производительность алгоритмов при конкретных условиях работы программы, описывает, как оптимизировать алгоритмы для конкретных ситуаций, учитывая особенности аппаратного и программного обеспечения.

* получение явных функций трудоёмкости;
* интервальный анализ функций;
* построение практических критериев качества алгоритмов;
* методики выбора рациональных алгоритмов;

1. **Цели и задачи теории алгоритмов**

* дальнейшая формализация понятия алгоритм и исследование формальных алгоритмических систем;
* формальное доказательство алгоритмической неразрешимости ряда задач;
* классификация задач, определение и исследование сложностных классов;
* асимптотический анализ сложности алгоритмов;
* исследование и анализ рекурсивных алгоритмов;
* получение явных функций трудоёмкости в целях сравнительного анализа алгоритмов;
* разработка критериев сравнительной оценки качества алгоритмов;

1. **Аспекты применения теории алгоритмов**

Практический аспект позволяет осуществить:

* рациональный выбор из известного множества алгоритмов решения данной задачи с учетом особенностей их применения;
* получение временны́х оценок решения сложных задач;
* получение достоверных оценок невозможности решения некоторой задачи за определенное время;
* разработку и совершенствование эффективных алгоритмов решения задач в области обработки информации на основе практического анализа;

Теоретический аспект позволяет:

* определить алгоритмическую разрешимость задачи;
* определить возможность сведения алгоритмически неразрешимых задач к задаче остановки машины Тьюринга;
* определить для алгоритмической разрешимой задачи факт принадлежности этой задачи к классу NP полных задач;
* оценить для члена класса NP полных задач временных затрат для получения точного решения для больших размерностей исходных данных;

**ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ**

1. **Принципы анализа алгоритмов**
2. Инвариантность проводимого анализа

*Инвариантность* - независимость от реализации на конкретной ЭВМ.

1. Анализу подвергаются алгоритмы, решающие одну и ту же проблему

Особенности “диалектов” языков высокого уровня (Pascal, C++, Java и др.) при этом не учитываются, поскольку структуры операторов управления функционально одинаковы.

1. Алгоритмы анализируются на классах данных

При анализе алгоритма необходимо учитывать характеристики входных данных, такие как объем, распределение, формат и т.д. Различные классы данных могут требовать различных алгоритмов для решения одной и той же проблемы. Если классы построены правильно, то характеристики алгоритма при работе на классе будут неизменны.

1. Учёт ресурса типа “память”

При анализе алгоритма необходимо учитывать затраты ресурсов, таких как память, процессорное время, сетевой трафик и т.д. Оптимальный алгоритм должен использовать минимальное количество ресурсов при выполнении задачи.

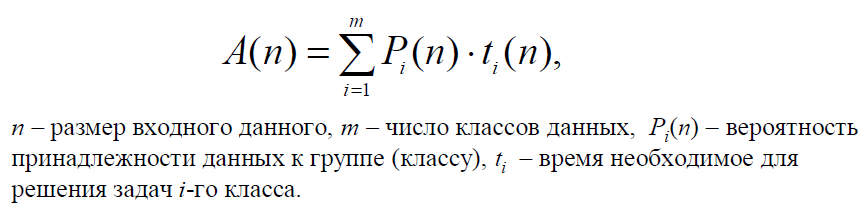
Оптимизация по занимаемой программой памяти и по быстродействию программы.

1. Поиск наилучшего, наихудшего и среднего случаев

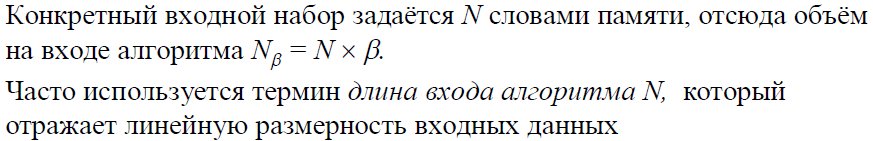
*Наилучший случай* – ситуация, для которой обработка некоторого класса данных даёт самый лучший показатель в рамках выбранной оценки. Обычно тривиален.

*Наихудший случай* – ситуация, для которой обработка некоторого класса данных даёт самый плохой показатель в рамках выбранной оценки. Позволяет определить максимальное время работы алгоритма.

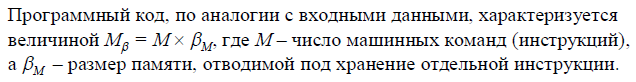
*Средний случай* – результат статистической обработки показателей, полученных для разны х классов данных.



1. **Допущения, принятые при проведении анализа**
2. В качестве вычислительного устройства, реализующего алгоритм, принимается абстрактная машина с процессором фон-Неймановской архитектуры, которая поддерживает произвольный доступ к памяти и множество элементарных операций, соответствующий набору конструкций языков “высокого уровня”;
3. Каждая команда выполняется не более соответствующего фиксированного времени;
4. Исходные данные представляются в виде машинных слов по β (байт/бит) каждое.



1. Программный код представляется последовательностью машинных слов;

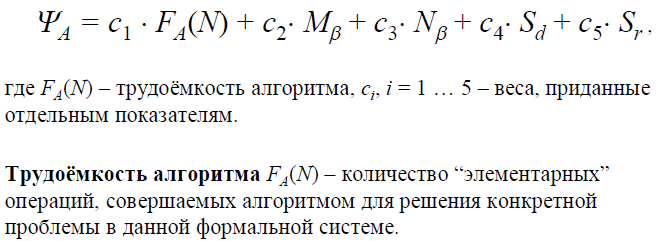


1. Факультативно (опционально) учитываются дополнительные ресурсы:

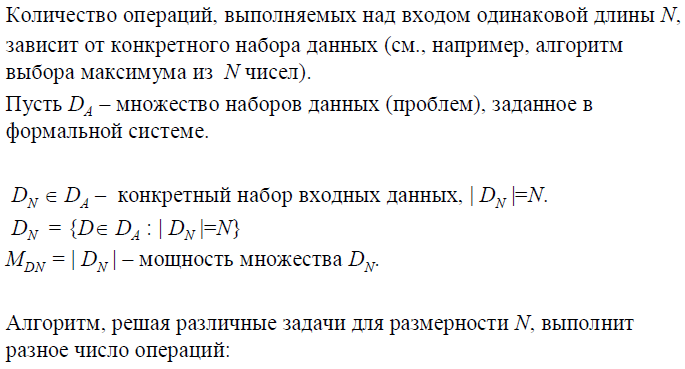
• – объём памяти для хранения промежуточных данных;

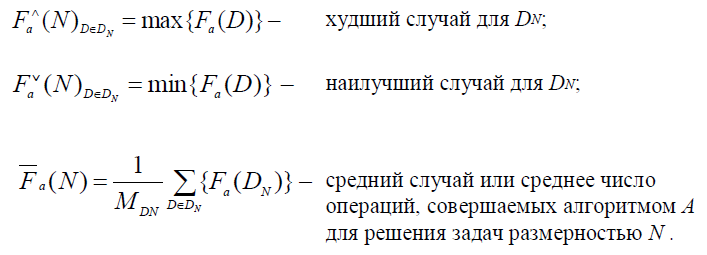
• – объём памяти, необходимой для организации вычислительного процесса (стеки, память для организации рекурсивных вызовов и возвратов, область буферов для осуществления операций ввода вывода).

1. Комплексный показатель оценки алгоритма



1. **Формальная классификация входных данных**





1. **Функции трудоёмкости алгоритмов**

Функции трудоёмкости есть результаты аппроксимации зависимостей трудоёмкости от длины входа алгоритма. Классификация функций трудоёмкости организована по виду входа:

1. Количественно-зависимые по трудоёмкости алгоритмы (N)

Для данной категории алгоритмов функция трудоёмкости зависит только от объёма входных данных, а не от их конкретных значений.



1. Параметрически-зависимые по трудоёмкости алгоритмы (PR)

Размерность, как правило, фиксирована, а трудоёмкость определяется конкретными значениями компонентов вектора данных. Это, например алгоритмы расчётов математических функций на базе степенных рядов с заданной точностью.



n – длина ввода, m – число параметров, оказывающих влияние на трудоёмкость.

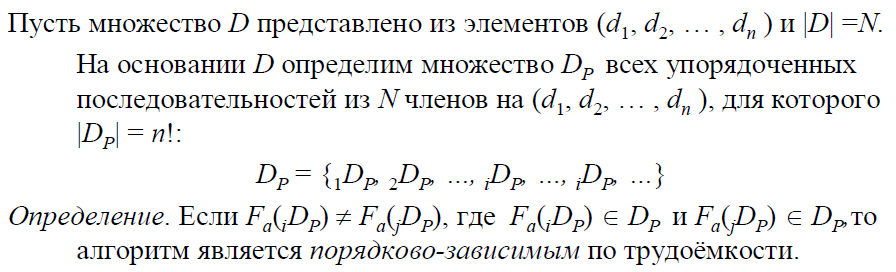
1. Количественно-параметрические по трудоёмкости алгоритмы (NPR)

Алгоритмы данного класса имеют функцию трудоёмкости, которая одновременно зависит как от количества (объёма) данных, так и от конкретных значений элементов данных.



1. Порядково-зависимые по трудоёмкости алгоритмы

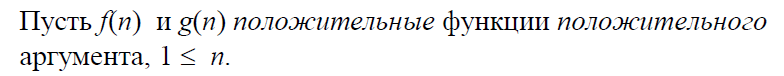
Для данных алгоритмов функция трудоёмкости определяется порядком следования данных в потоке.



1. **Асимптотический анализ. Виды оценок**

*Асимптотический анализ* применяется для оценки (прогноза) количества операций, лежащих в основе определения сложности.

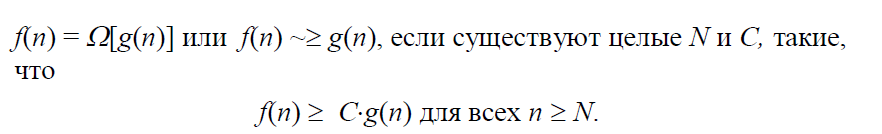
Целью является оценка скорости роста числа операций при возрастании объёма водных данных.



Виды оценок:

1. Оценка снизу Ω:

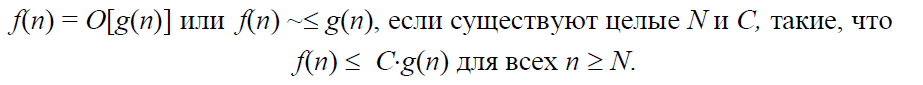
Определён класс функций, которые растут не медленнее, чем g(n) с точностью до постоянного множителя, начиная с некоторого объёма данных N.



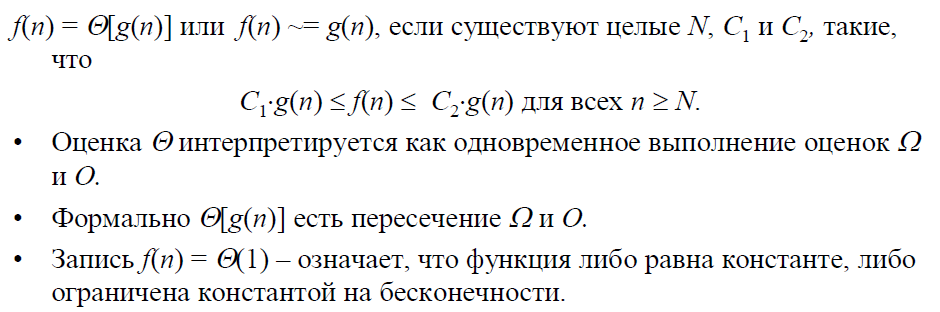
1. Оценка сверху Ο:

Оценка определяет класс функций, которые растут не быстрее, чем функция g(n) с точностью до некоторого постоянного множителя;

Функция g(n) *мажорирует* f(n).



1. Оценка Θ:



*Мера ρ[f(n), g(n)]* – количественная оценка расхождения функций f(n) и g(n).

*Порог значимости h* – величина, мерой расхождения ρ[f(n), g(n)] не более которой можно пренебречь.

1. **Асимптотический анализ. Методы**
2. Метод раскрутки

* Метод применяют, когда анализируемая функция удовлетворяет некому неявному уравнению.
* Последовательно подставляя в уравнение асимптотическое приближение к функции, полученное на предыдущем шаге, добиваемся улучшения асимптотической оценки.

1. Метод расчленения

* Применяется к суммам и интегралам.
* Сумма многокомпонентная.
* Ни один из компонентов не является пренебрежимым на всей области суммирования или интегрирования

**ВРЕМЕННЫЕ ОЦЕНКИ ТРУДОЁМКОСТИ АЛГОРИТМОВ**

1. **Постановка задачи временной оценки трудоёмкости алгоритма**

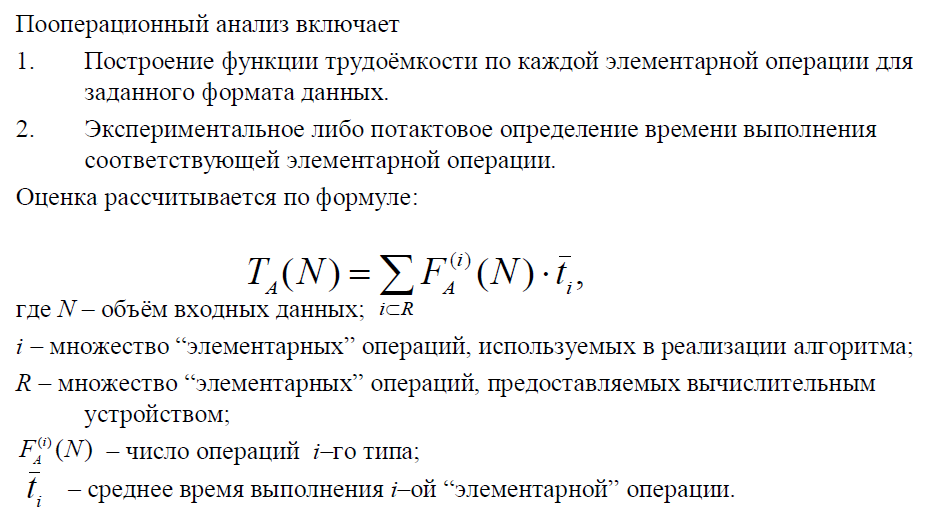
Задан: Алгоритм А, обладающий трудоёмкостью FA(DA).

Необходимо: оценить время машинной реализации ТA(DA) алгоритма А.

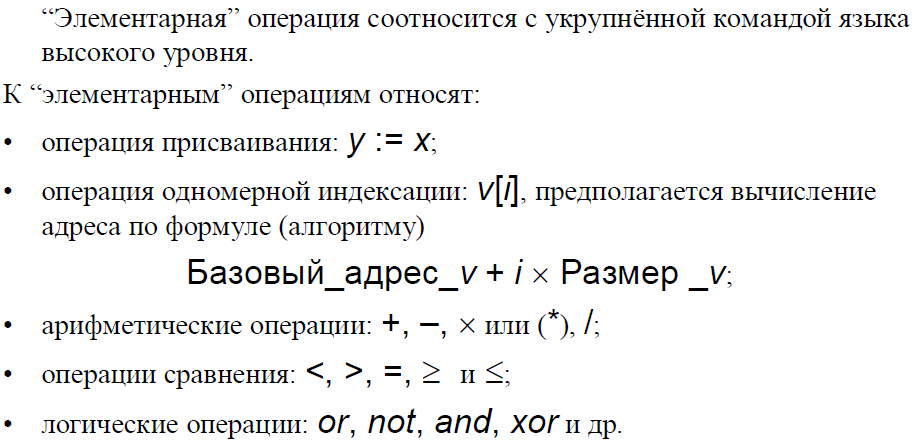
Временная оценка определяется рядом факторов:

* Состав команд процессора, необходимых для отображения отдельных операций алгоритма;
* Количество этих команд;
* Время выполнения отдельных типов команд;
* Особенности формальной системы записи алгоритма на языке программирования;
* Особенности трансляции исходной программы в машинный код;
* Различие во времени выполнения разнотипных команд процессора;
* Различие во времени выполнения однотипных команд для разных форматов (типов) данных;
* Различия, обусловленные типами адресации;
* Наличие у процессора особенностей архитектуры, способствующих ускорению вычислений;

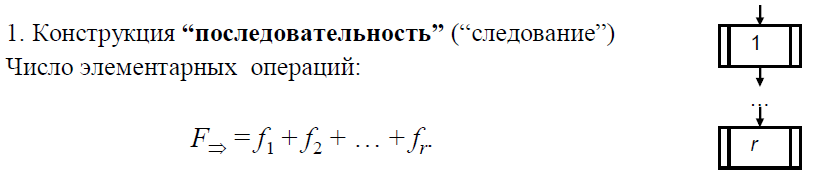
1. **Пооперационный анализ**



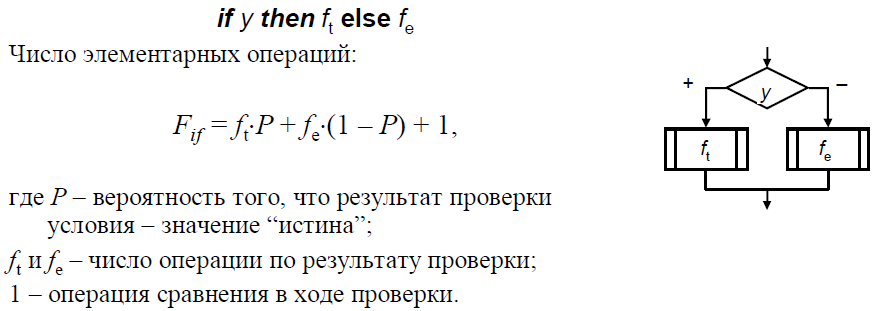
1. **Понятие “элементарной” операции**



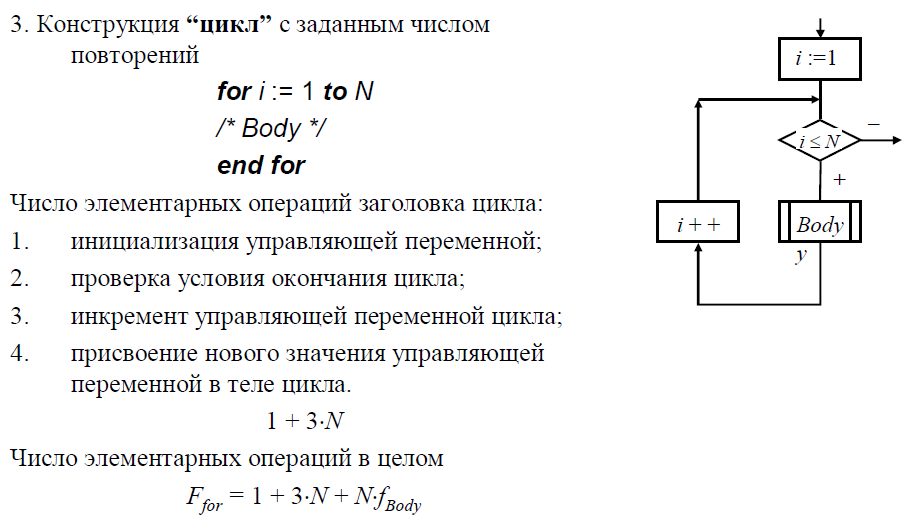
1. **Алгоритмические конструкции**
2. Последовательность



1. Ветвление



1. Цикл

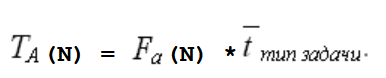


1. **Метод Гиббсона**

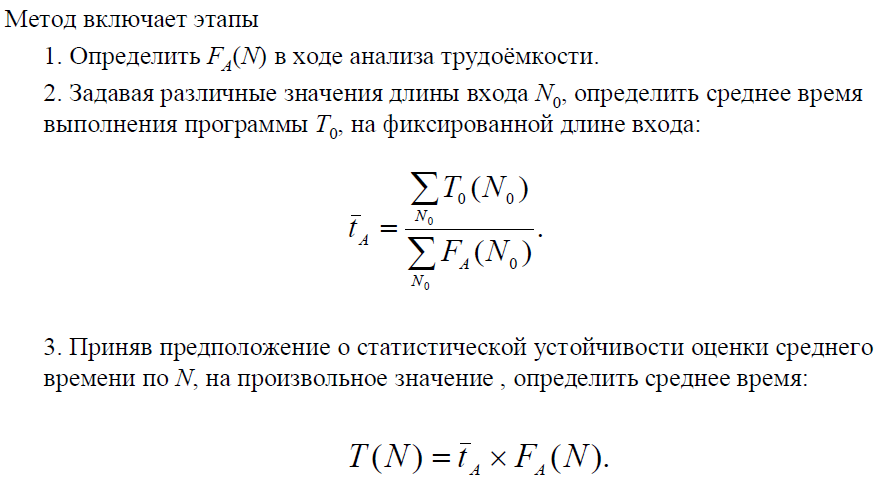
Метод предполагает проведение совокупного анализа по трудоемкости и переход к временным оценкам на основе принадлежности решаемой задачи к одному из следующих типов:

* задачи научно-технического характера с преобладанием операций с операндами вещественного типа;
* задачи дискретной математики с преобладанием операций с операндами целого типа;
* задачи баз данных с преобладанием операций с операндами строкового типа;

Далее на основе анализа множества реальных программ для решения соответствующих типов задач определяется частотная встречаемость операций, создаются соответствующие тестовые программы, и определяется среднее время на операцию в данном типе задач – *тип задачи*.

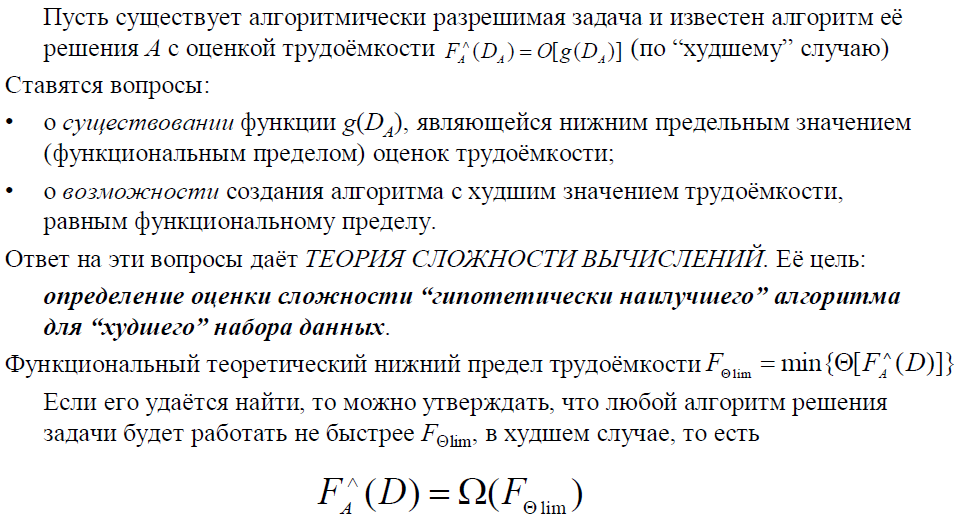


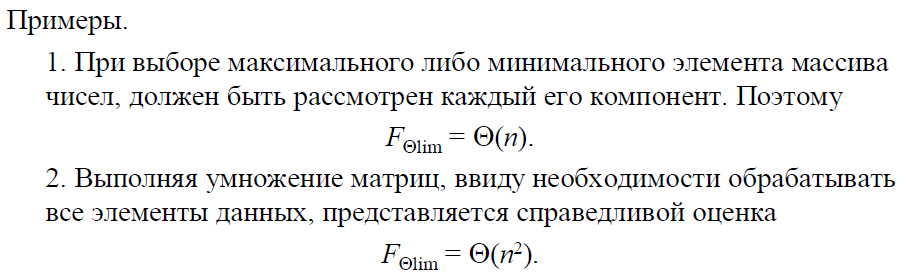
1. **Метод прямого определения среднего времени**



**СЛОЖНОСТНЫЕ КЛАССЫ ЗАДАЧ**

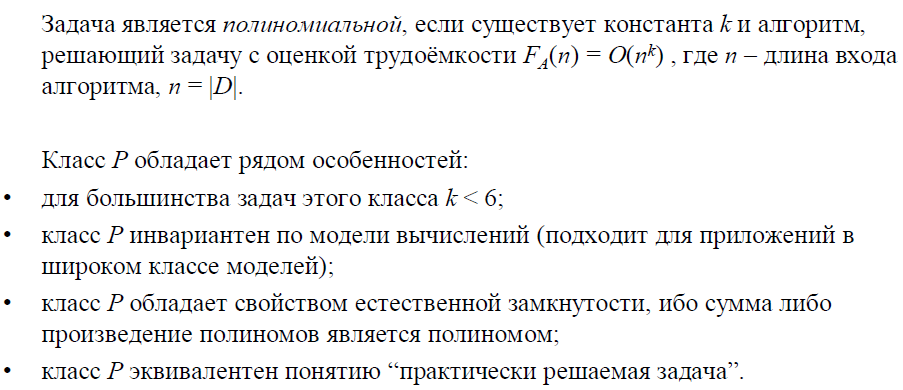
1. **Постановка задачи классификации алгоритмов**





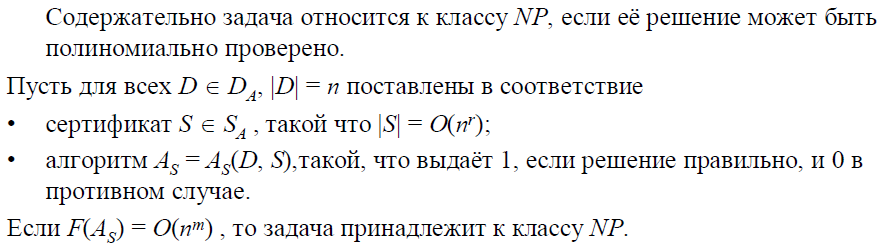
1. **Классификация сложности**
2. Класс Р – задачи с полиномиальной сложностью

Это класс задач, которые могут быть решены за полиномиальное время на детерминированной машине Тьюринга. Такие задачи считаются эффективно решаемыми.



1. Класс NP – полиномиально проверяемые задачи

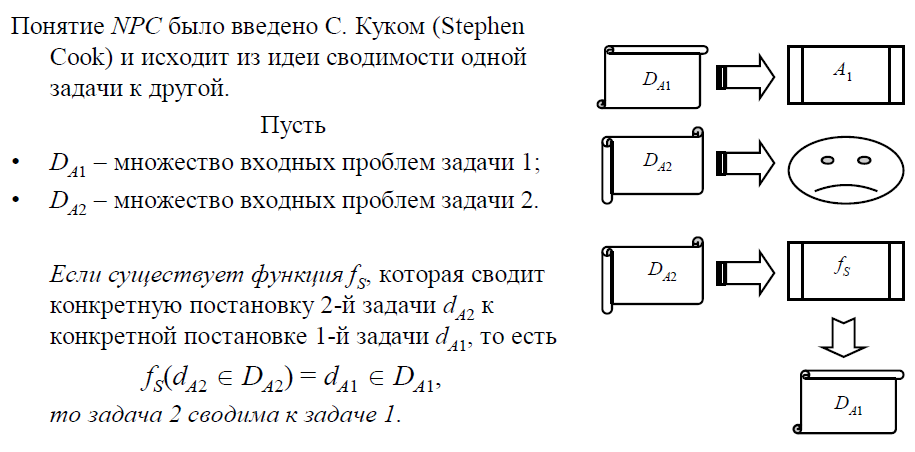
Это класс задач, для которых можно проверить правильность решения за полиномиальное время, но нет известного алгоритма для их решения за полиномиальное время. Эти задачи считаются труднорешаемыми, но возможно их приближенное решение.

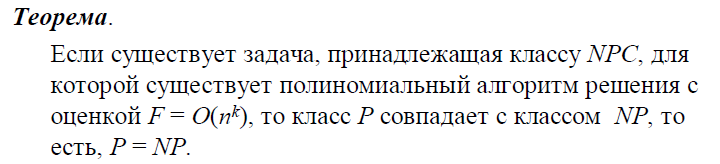


Доказательства совпадения или несовпадения классов P и NP отсутствуют. Существует предположение, что P является собственным подмножеством класса NP.

1. Класс NP-полных задач (NPC: NP-complete)

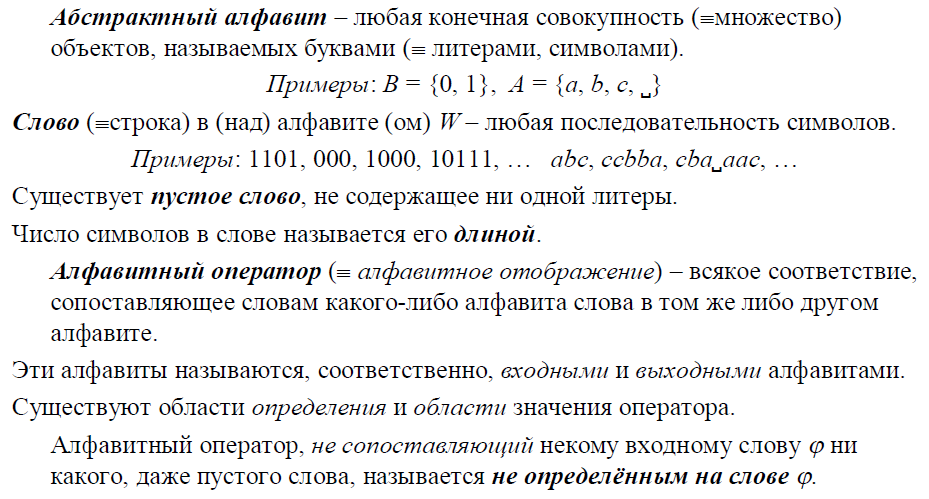
Это самые труднорешаемые задачи в NP-классе. Для них не существует известных полиномиальных алгоритмов решения, но известны алгоритмы с экспоненциальной сложностью. Если одна NP-сложная задача может быть решена за полиномиальное время, то все задачи из класса NP могут быть решены за полиномиальное время.

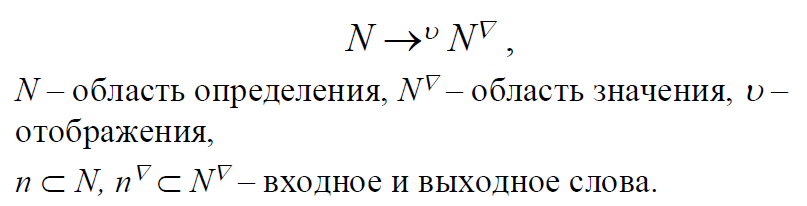


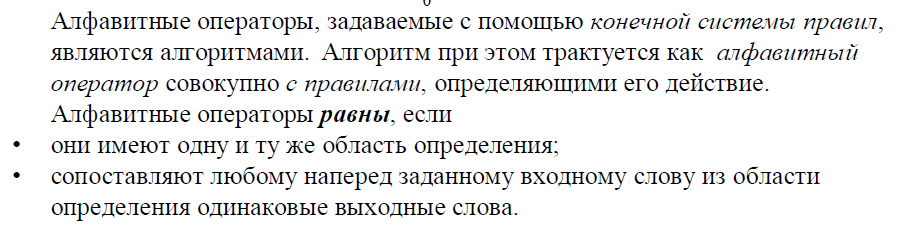


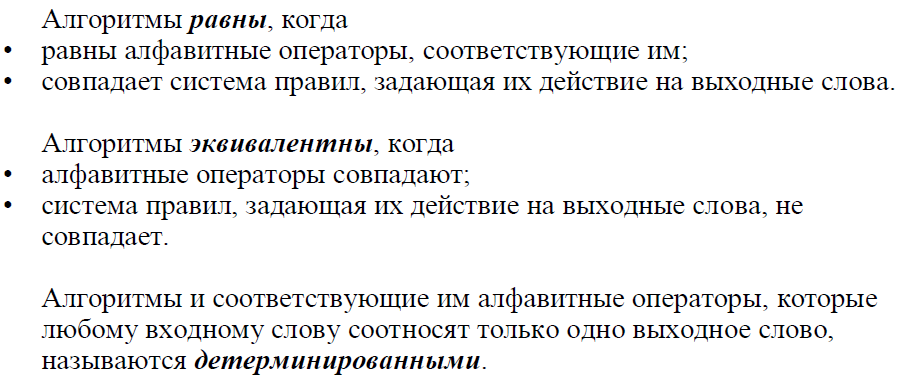
**АЛГОРИТМЫ И ИСЧИСЛЕНИЯ**

1. **Алфавитный оператор**

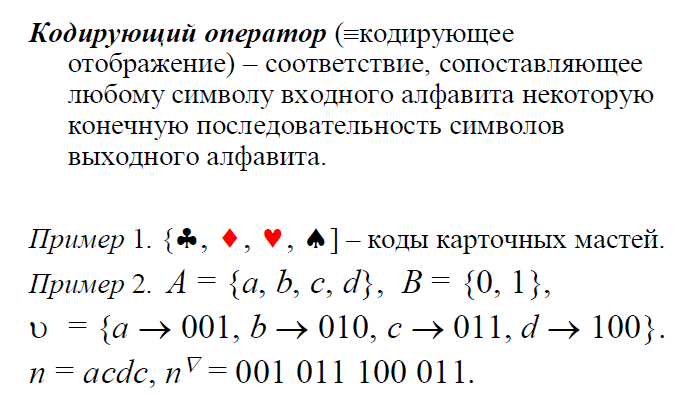


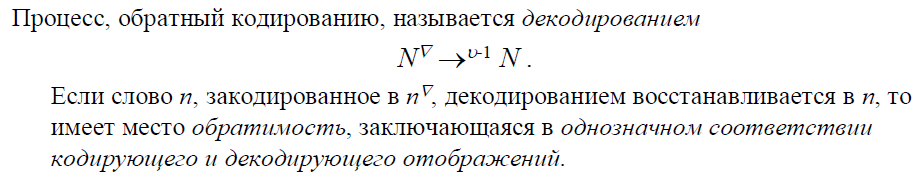






1. **Кодирующий оператор**





1. **Рекурсия в вычислениях**

*Рекурсия* – способ задания функции, при котором значение определяемой функции для произвольных значений аргументов выражается известным образом через значения определяемой функции для меньших значений аргументов.

* приём, сводящий общую задачу к более простой задаче того же класса;

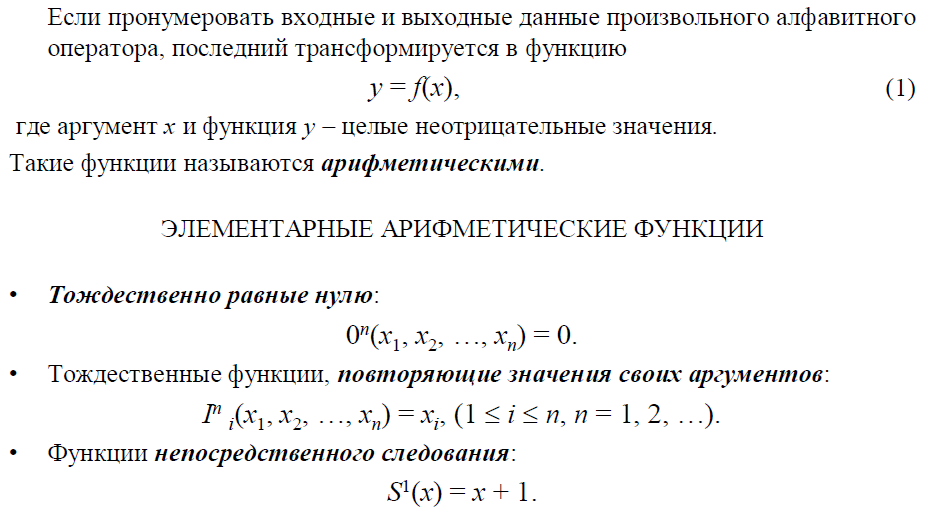
*Индукция* – разновидность рекурсии в методологии поведения доказательств.

Если удаётся показать, что функция, решающая некоторую задачу, не может быть рекурсивной, то задача не разрешима.

Применение рекурсивных функций в теории алгоритмов основано на предложении перенумеровать слова в произвольном алфавите натуральными числами в порядке возрастания.

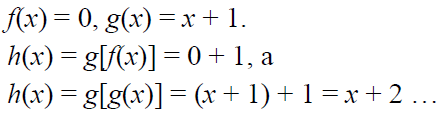
В качестве способа такой нумерации, слова располагаются в порядке возрастания их длин, а слова одинаковой длины упорядочиваются в алфавитном порядке.

1. **Арифметические функции**



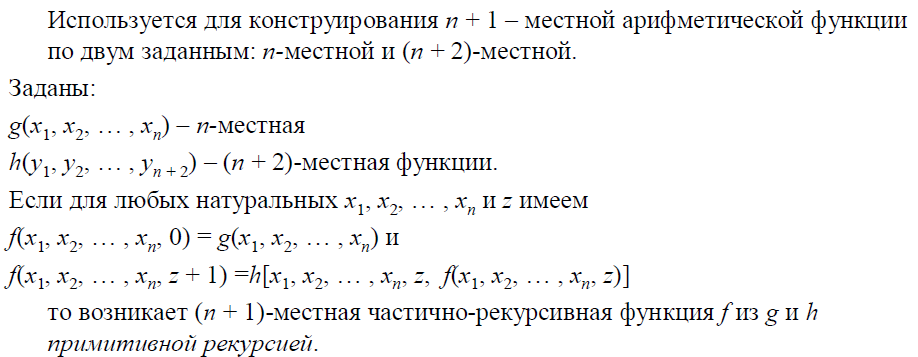
1. **Конструктивные приёмы**
2. Операция суперпозиции функций

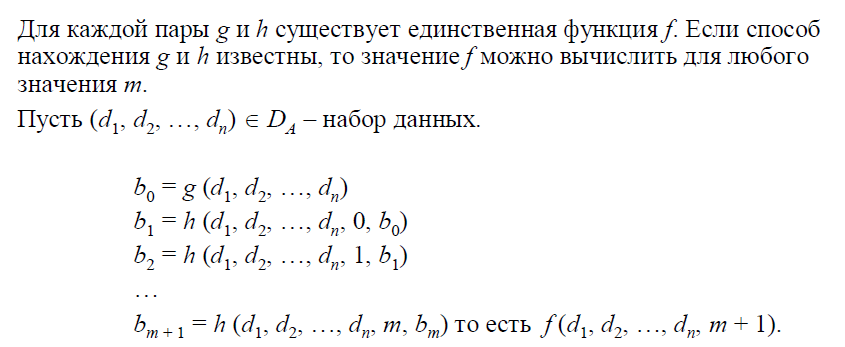
Сущность операции: подстановка одних арифметических функций вместо аргументов других.



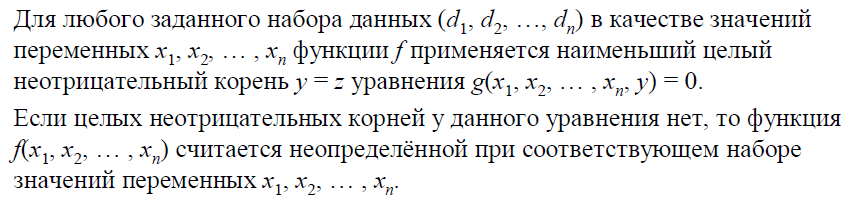
1. Операция примитивной рекурсии

*Примитивно-рекурсивными* называются функции, которые могут быть построены из элементарных арифметических операций с помощью операций суперпозиции и примитивной рекурсии, примененных произвольное число раз и в произвольной последовательности

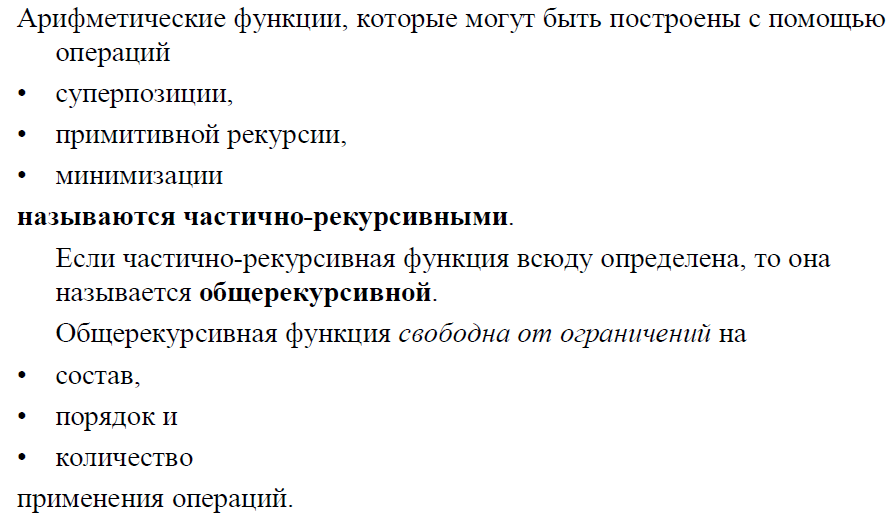


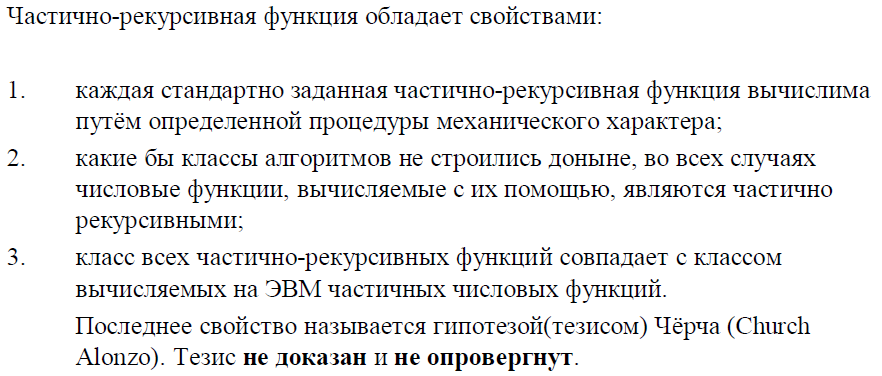


1. Операция наименьшего корня (минимизация)



1. **Частично-рекурсивные функции**





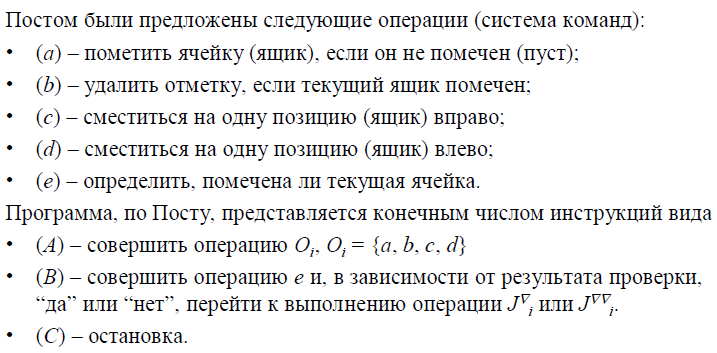
**ВЫЧИСЛИТЕЛЬНЫЕ МОДЕЛИ**

1. **Финитный 1-процесс**

Допущения при формулировке задачи по Посту:

* имеем общую проблему, состоящую из множества конкретных проблем;
* решением общей проблемы будет решение всех частных;
* проблемы сформулированы в пространстве символов;
* имеется конечный набор инструментов (инструкций клерку), зафиксированный и не изменяемый для всех проблем;

Конкретная проблема задается пометкой конечного количества ячеек, при этом, очевидно, что любая конфигурация начинается и заканчивается помеченным ящиком. После применения к конкретной проблеме некоторого набора инструкций решение представляется так же в виде набора помеченных и непомеченных ящиков.



Программа (набор инструкций в терминах Поста) является одной и той же для всех конкретных проблем, поэтому соотнесена с общей проблемой – таким образом, Пост формулирует требование универсальности.

Далее Пост вводит следующие понятия:

* набор инструкций применим к общей проблеме, если для каждой конкретной проблемы не возникает *коллизий* в инструкциях a и b (никогда программа не стирает метку в пустом ящике и не устанавливает метку в помеченном ящике);
* набор инструкций заканчивается (за конечное количество инструкций), если выполняется инструкция (C);
* набор инструкций задаёт *финитный 1–процесс*, если набор применим и конечен для каждой конкретной проблемы;
* финитный 1–процесс для общей проблемы есть *1–решение*, если ответ для каждой конкретной проблемы правильный;

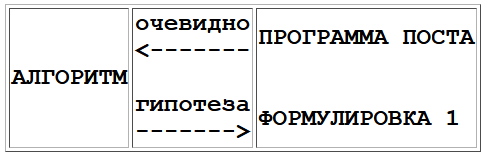
Пусть между классом целых чисел и классом конкретных проблем установлено взаимно-однозначное соответствие. Общую проблему будем называть *1-заданной*, если есть финитный 1-процесс, который в результате применения к положительным числам, выдаёт номера проблем, образующих общую проблему.

1. **Формулировка гипотезы Поста**

Алгоритмы 1 и 2 называются *равносильными*, если к любому данному из общей их совокупности они либо оба применимы и дают единый результат, либо оба не применимы.

Гипотеза:

Каждый алгоритм, все результаты которого суть числа, а областью исходных данных служат N, Nk или N∞, *равносилен* алгоритму с такой же совокупностью возможных исходных данных, задаваемой некоторой программой машины Поста.

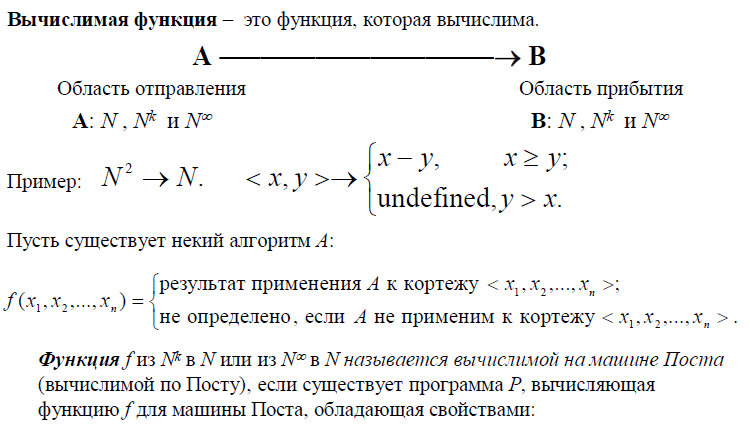
*Дополнительно:* если общая проблема *1-задана* и *1-разрешима*, то, соединяя наборы инструкций по заданию проблемы и её решению мы получаем ответ по номеру проблемы – это и есть *формулировка 1* по Посту.

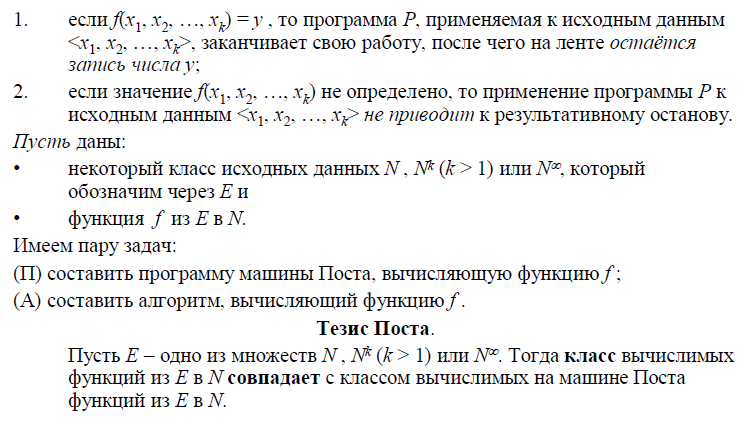
Гипотеза Поста состоит в том, что любые более широкие формулировки и интерпретации конкретных проблем сводимы к формулировке 1.

Следовательно, если гипотеза верна, то любые другие формальные определения, задающие некоторый класс алгоритмов, эквивалентны классу алгоритмов, заданных формулировкой 1 Эмиля Поста.

Обоснование этой гипотезы происходит сегодня не путем строго математического доказательства, а на пути эксперимента — всякий раз, когда нам указывают алгоритм, его можно перевести в форму программы машины Поста, приводящей к тому же результату.

1. **Вычислимые функции и тезис Поста**

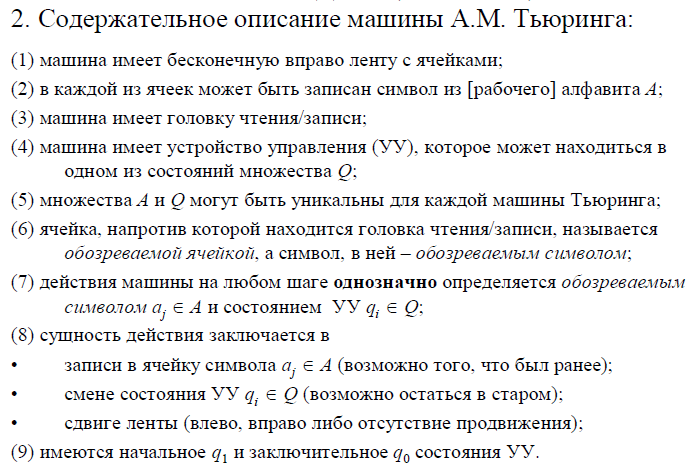


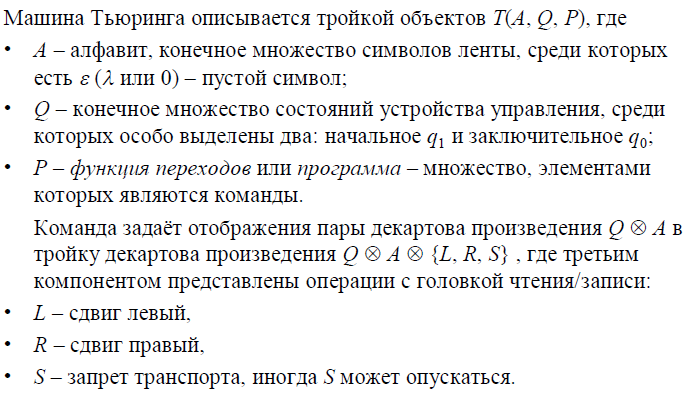


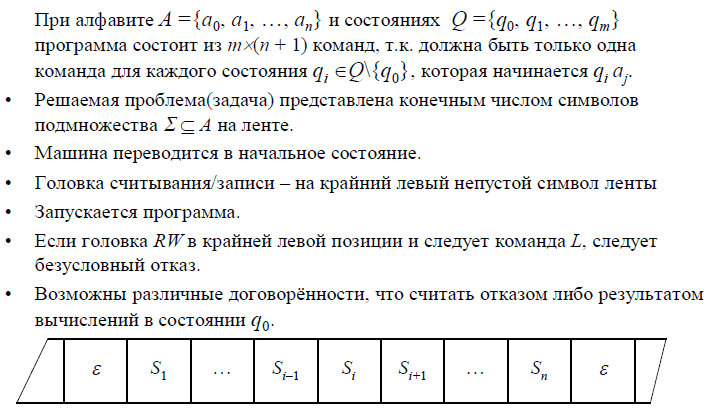
1. **Постулат Поста**

Задача на составление программы, приводящей от исходного данного к результирующему числу (и не приводящей ни к какому результату, если результирующего числа не существует), тогда и только тогда имеет решение, когда имеется какой-нибудь общий способ, позволяющий по произвольному исходному данному выписать результирующее число и не выдающий никакого – заведомо ложного – в этом случае результата, коль скоро результирующего числа не существует.

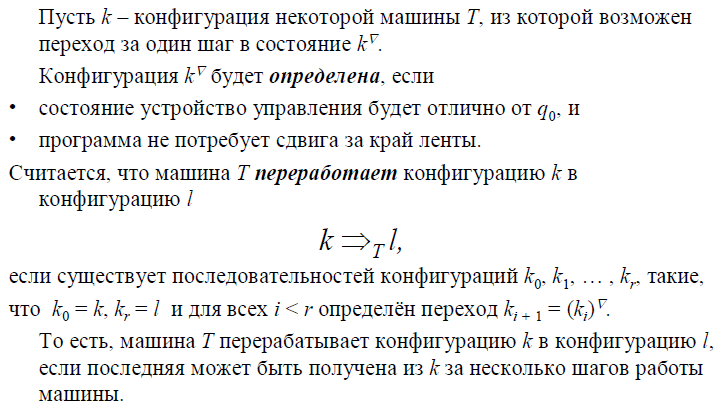
1. **Формальное описание машины Тьюринга**



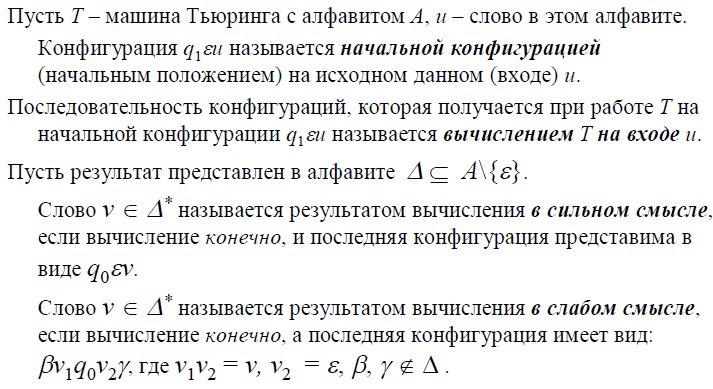


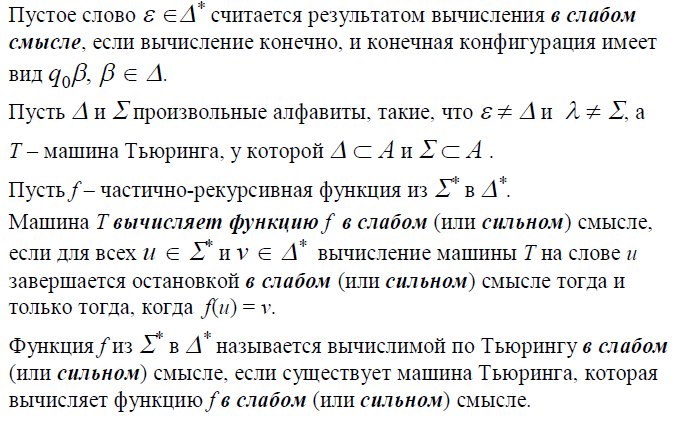


Конфигурация (машинное слово, мгновенное описание) – тройка, состоящая из состояния qr, положения головки на ленте, содержимого ленты;

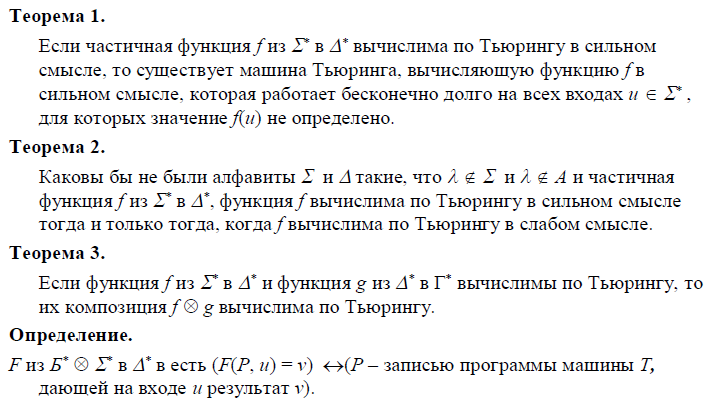


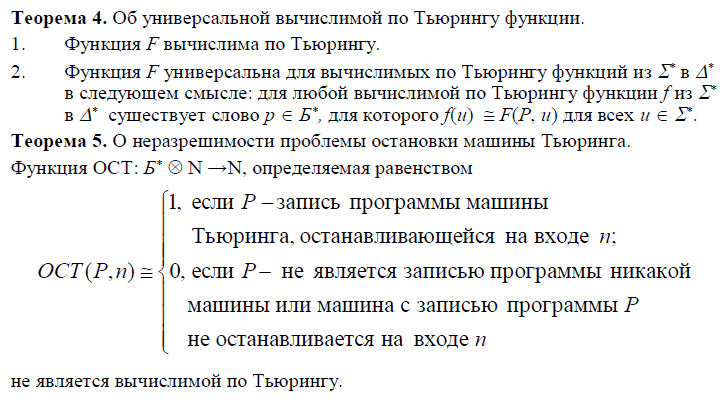
1. **Вычислимость в сильном и слабом смыслах**





1. **Теоремы, связанные с вычислительной моделью Тьюринга**





1. **Тезис Тьюринга**

Каждая функция, для вычисления значений которой существует будь-какой алгоритм, оказывается вычислимой посредством некоторой машины Тьюринга.

Тезис:

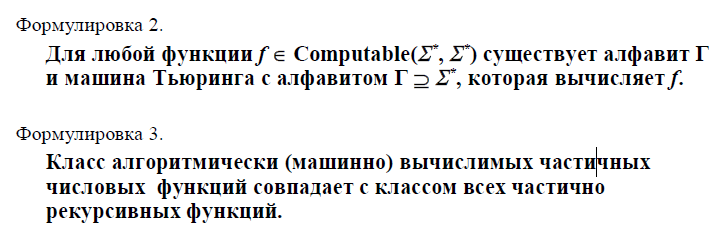
Для нахождения значения функции, заданной в некотором алфавите, тогда и только тогда существует какой-нибудь алгоритм, когда функция является вычислимой по Тьюрингу, т.е. когда она может быть вычислена на подходящей машине Тьюринга.

1. **Тезис Чёрча**

Числовая функция тогда и только тогда алгоритмически (машинно) вычислима, когда она частично рекурсивна *(Формулировка 1)*.

Пост и Тьюринг одновременно с Чёрчем и не зависимо друг от друга утверждают, что класс всюду определённых функций, вычислимых в определённой модели, совпадает с классом всюду определённых вычислимых функций (для фиксированных ансамблей).

Тезис Чёрча, в узком смысле, утверждает, что всякая вычислимая функция с натуральными аргументами и значениями частично рекурсивна.

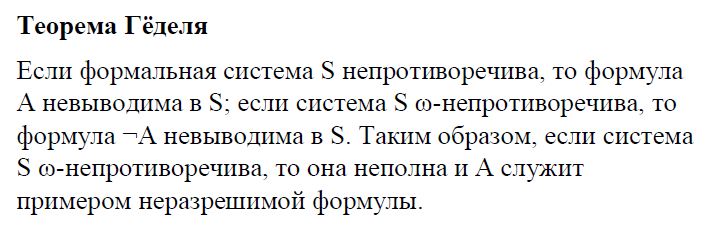


1. **Неразрешимые алгоритмические проблемы**

Теорема: не существует алгоритма (машины Тьюринга), позволяющего по описанию произвольного алгоритма и его исходных данных (и алгоритм и данные заданы символами на ленте машины Тьюринга) определить, останавливается ли этот алгоритм на этих данных или работает бесконечно.

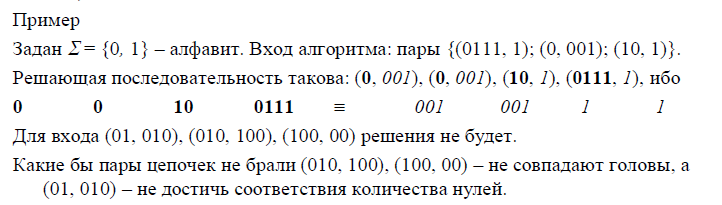
Причины:

* Отсутствие общего метода решения задачи
* Информационная неопределённость задачи
* Логическая неразрешимость в смысле теоремы Гёделя о неполноте



1. **Понятие частичного алгоритма**

*Решающая последовательность* – конечная последовательность пар, (не обязательно из различных xi и yi), такая, что цепочка, составленная из левых компонентов пар, совпадает с последовательностью, составленной из правых подцепочек.



Для решения задачи можно сконструировать *частичный алгоритм*, строящий всевозможные упорядоченные возможные последовательности, проверяющий для каждой генерации условия решения.

Частичный алгоритм возможно, но не обязательно (!), находит решение проблемы.

Если существует решающая последовательность, то решение может быть получено за конечное число шагов.

Поскольку имеется частичный алгоритм, то задача называется *частично-разрешимой*.

Так как общий метод определения отсутствия решающей последовательности не может быть указан, следовательно, задача сведена к *проблеме останова*, следовательно, алгоритмически не разрешима.

Алгоритм нахождения эквивалентности и тотальности не имеют даже частичных алгоритмов

1. **Алгоритм преобразования структурных схем алгоритмов Ашкрофта-Манны**

